# **Step By Step Conducting the Interview**

1. Ask Refining Questions

* Requirements that the clients need directly—for example, the ability to send messages in near real-time to friends.
* Requirements that are needed indirectly—for example, messaging service performance shouldn’t degrade with increasing user load.

2. Handle the Data

* What’s the size of the data right now?
* At what rate is the data expected to grow over time?
* How will the data be consumed by other subsystems or end users?
* Is the data read-heavy or write-heavy?
* Do we need strict consistency of data, or will eventual consistency work?
* What’s the durability target of the data?
* What privacy and regulatory requirements do we require for storing or transmitting user data?

3. Discuss the Components

* Front-end components, load balancers, caches, databases, firewalls, and CDNs are just some examples of system components.

4. Discuss Trade-offs

* Different components have different pros and cons. We’ll need to carefully weigh what works for us.
* Different choices have different costs in terms of money and technical complexity. We need to efficiently utilize our resources.
* Every design has its weaknesses. As designers, we should be aware of all of them, and we should have a follow-up plan to tackle them.

# **What is system design**

System design is the process of defining components and their integration, APIs, and data models to build large-scale systems that meet a specified set of functional and non-functional requirements.

Diagram, shape

Description automatically generated

System design aims to build systems that are reliable, effective, and maintainable, among other characteristics.

* **Reliable** systems handle faults, failures, and errors.
* **Effective** systems meet all user needs and business requirements.
* **Maintainable** systems are flexible and easy to scale up or down. The ability to add new features also comes under the umbrella of maintainability.

# **Why Are Abstractions Important?**

Abstraction is the art of obfuscating details that we don’t need. It allows us to concentrate on the big picture. Looking at the big picture is vital because it hides the inner complexities, thus giving us a broader understanding of our set goals and staying focused on them. The following illustration is an example of abstraction. The developers use a lot of libraries to develop the big systems. If they start building the libraries, they won’t finish their work.